Packaging

•••

Advanced Track: Lecture 2

Daniel Kessler (adapted from mdcha)

Administrivia

- Lab 1 was due Saturday
  - Late deadline is in 2 weeks, 2/19
  - Make sure to complete labs before the late deadline for credit
- VMs have been distributed!
  - Try logging in now: `ssh <username>@<username>.decal.xcf.sh`
  - Email us at decal@ocf.berkeley.edu if you didn't get one
  - You'll need it to complete lab a2 so make sure it works!!
- Lab 2 due this week

About me

Outline

- Why do we have packages?
- What is a package?
- Repositories vs App Stores
- Package internals
- Repositories

How do you install software?

- What files do you need? How do you get them?
- How much work does the end user need to do?
- How much work does the developer need to do?
- Anyone else involved in distributing software?
- How do distribute updates?
- Any security concerns?

Naïve solution

Download a set of files

- compiled binaries,
- documentation,
- Config files
- license(s)
- Internationalization / localization files

Any problems?

Naïve solution and its issues

- Which files do I need to download?
- Where do I put each file?
- Configuration issues?
- Will it work on my system?
- Updates?

Tarball solution

Download a tarball from the application developer (“upstream”):

- Should the source code be compiled or pre-compiled?
- What if the software depends on other software being installed?
- Each Linux distribution is different: does the application developer need to learn the quirks of each?
- How do we get updates?
- How can users trust that updates won’t break or be disruptive?

Packaging!

A centralized system for managing package metadata

- Each Linux distribution maintains **repositories** , a set of packages available to download
- Volunteers (**package maintainers**), on behalf of distributions, are responsible for packaging developers’ software
- Each distribution has processes and policies for how software should be packaged

Repositories

A **package** is a single piece of software to be distributed.

- htop is a package. Each of its dependencies are packages

A **repository** is a set of packages, usually served from the Internet.

- Your machine is configured to fetch packages from repositories

Examples of package repositories:

- Debian stable (all the packages available to install on a default Debian stable install)
- Debian security updates
- Debian backports
- Docker's custom repositories

Any given Linux machine is probably pulling from multiple repos!

Demo: installing a package

You may have seen or done this before...

What just happened (Con’t)

`apt install`:

a. Reads from the package lists
b. Finds out what dependencies the package needs
c. Checks what packages are already installed
d. Downloads the packages (if not installing a local .deb)
e. Unpacks them and copies the files over
f. Processes any remaining triggers
    i. Triggers are events such as scripts that run post-install
       An example is starting the application as a service

What about App Stores?

Linux distribution package repositories and app stores are both are centralized places to get software,

...but there are key differences.

Packages vs Apps: distributing updates

Packages:

- Maintainers vet updates
- Disruptive updates could get delayed
- Users who want to test newer versions can subscribe to newer tracks (Debian: “testing” or “unstable”)

Apps

- Developers push updates directly to users
- Canarying: developers might randomly select users to test new versions on
- Minimal bureaucracy/friction to pushing updates

Packages vs Apps: security & trust

Packages

- Users trust distribution to not package sneaky software
- Package maintainers patch out anti-features when necessary
- Some distros will include AppArmor/SELinux profiles as an extra security precaution

Apps

- Users hope developers won’t be sneaky
- Permissions, consent dialogs, and OS features (e.g. “x is reading your location”) place technical limitations
- Strong sandboxing/isolation

Packages vs Apps: dependencies

Packages

- Packages can depend on other packages (libraries)
- Dependencies can get updated independently of packages that depend on them
- Your distribution ensures that dependency upgrades don’t break anything

Apps

- Apps are sandboxed and isolated, can only depend on the underlying OS
- App developers are responsible for keeping dependencies up to date
- Developers are responsible for testing when dependencies are updated

Packages vs X

Other forms of software distribution (that we won’t discuss today):

- Webapps and the web platform
- Snaps/Flatpaks (kind of like “app stores” for desktop Linux)
- F-Droid (alternate app store for Android)
- Helm: packages for Kubernetes

Exercise to the reader: if you’re familiar with any of these, think about the trade-offs of **stability** , **dependencies** , **security** , **isolation** , etc

Packages vs Apps:

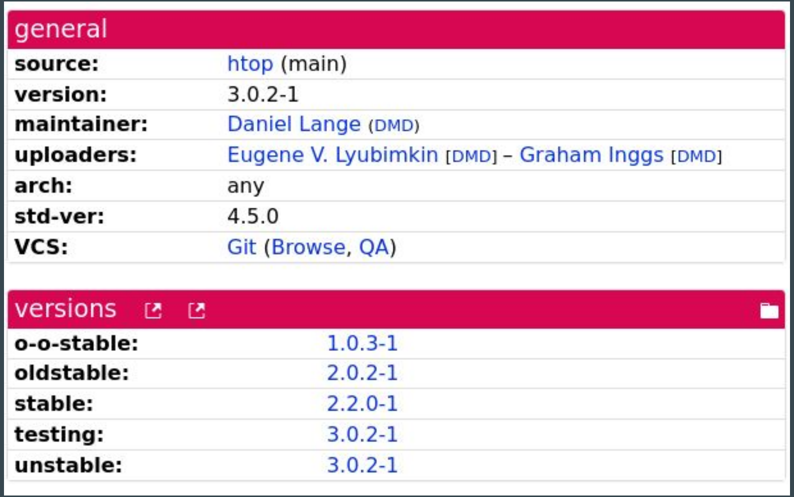
Packages

Apps

Why is my software so out of date?

- In a "periodic release" OS (like Debian), updates can get delayed to major version releases
    - New Debian releases happen every ~2 years
- In a "rolling release" OS, you might get updates faster, but at the cost of stability
    - e.g. Arch Linux tries to get new releases ASAP

Rolling distribution releases versus periodic releases are a tradeoff by Chris Siebenmann



general

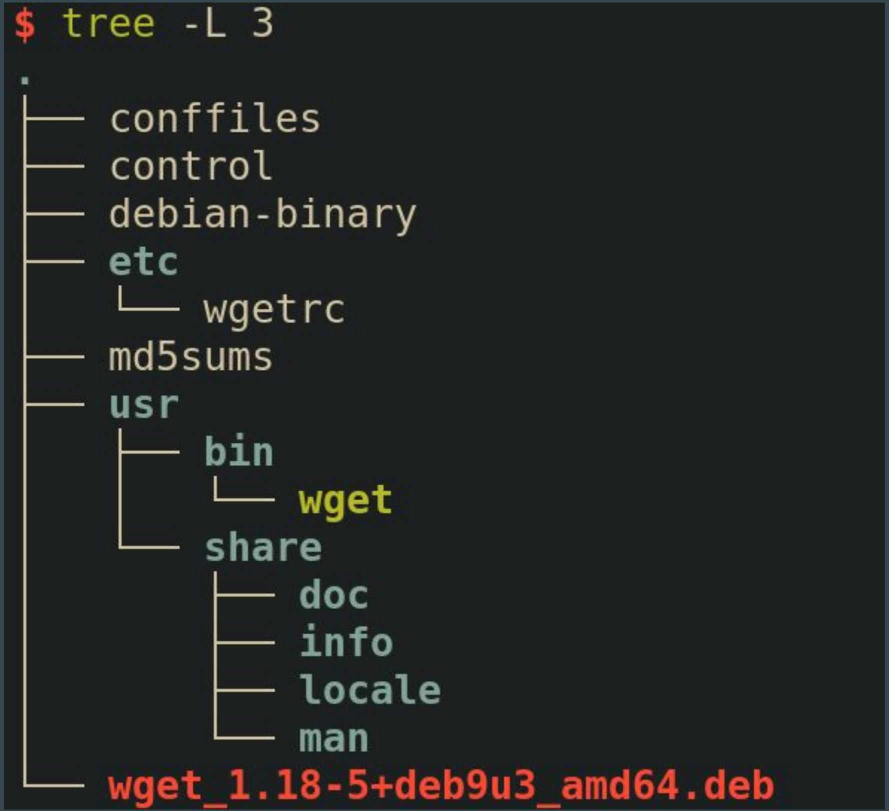
source:	htop (main)
version:	3.0.2-1
maintainer:	Daniel Lange (DMD)
uploaders:	Eugene V. Lyubimkin [DMD] – Graham Inggs [DMD]
arch:	any
std-ver:	4.5.0
VCS:	Git (Browse, QA)

versions

o-o-stable:	1.0.3-1
oldstable:	2.0.2-1
stable:	2.2.0-1
testing:	3.0.2-1
unstable:	3.0.2-1

source: https://tracker.debian.org/pkg/htop (accessed 2020-09-22)

Anatomy of a Debian package



```
$ tree -L 3
.
├── conffiles
├── control
├── debian-binary
├── etc
│   └── wgetrc
├── md5sums
├── usr
│   ├── bin
│   │   └── wget
│   └── share
│       ├── doc
│       ├── info
│       ├── locale
│       └── man
└── wget_1.18-5+deb9u3_amd64.deb
```

Anatomy of a Debian package

- control: Package metadata
- Size of package
- Package version
  - For package updates
- Dependencies of this package

Anatomy of a Debian Package

- /usr/bin/: Executable(s) the package provides
- Added to your $PATH
- /usr/share:
    - Documentation
    - Manpages
    - locales
- /etc: global configuration files
- md5sums: verify integrity of all files

Anatomy of a Debian Package

- How can i view a package deb?
- .deb files are just ar archives
- apt download wget && ar x wget*_amd64.deb

Demo: Spelunking in a .deb

Demo

- Downloading a .deb w/ `apt download wget`
- Breaking it open with `ar`
- Examining the control file
- Examining the data file
- Being lazy and using `apt show`

Dependencies

- Hard problem. Graph theory at work. Yay CS70!
- Why? Attempt at reducing work. Could you imagine a world where you had to write every single piece of code you used?
- Pre-depends
- Conflict vs Breaks
- Recommends, suggests, enhances
- Provides: Virtual packages

Package Management

- Every distribution uses a different package manager
  - ex. Arch `pacman`, RHEL `yum`, MacOS `brew`
- Debian uses `dpkg` as the backend to actually install the packages, but most people use `apt`

Package Management

ocf note: unqualified URLs get .ocf.berkeley.edu appended
so http://mirrors/ is really http://mirrors.ocf.berkeley.edu/

How does Debian know which package to use?

/etc/apt/sources.list and /etc/apt/sources.list.d/*

```
deb http://mirrors/debian/ stretch-backports main contrib non-free
deb http://mirrors/debian-security/ stretch/updates main contrib non-free
deb-src http://mirrors/debian-security/ stretch/updates main contrib non-free
deb http://mirrors/debian/ stretch-updates main contrib non-free
deb-src http://mirrors/debian/ stretch-updates main contrib non-free
deb http://mirrors/debian/ stretch main contrib non-free
deb-src http://mirrors/debian/ stretch main contrib non-free
# OCF
deb http://apt/ stretch-backports main
deb-src http://apt/ stretch-backports main
deb http://apt/ stretch main
deb-src http://apt/ stretch main
deb http://mirrors/puppetlabs/apt/ stretch puppet
```

Breaking down a line

```
deb http://mirrors/debian/ stretch-backports
main contrib non-free
```

- deb is binary package source. deb-src indicates src packages.
- http://... describes the location
- stretch-backports means that this is for Jessie stretch and from a backport repository

Breaking down a line

```
deb http://mirrors/debian/ stretch-backports
main contrib non-free
```

- main means that the repo has packages licensed under Debian Free Software Guidelines (DFSG)
- contrib repos have packages licensed under DSFG but require non-free dependencies
- non-free repos have packages that do not comply with DSFG

Demo: installing from backports

Demo: installing from backports

- https://backports.debian.org/Instructions/
- Different repositories have different versions of the same package!
- Debian will pick the one with the highest priority, not necessarily highest version
- Can always use `apt policy` to see which repository your packages are coming from (even if multiple repositories have the same package)

Non-standard repos

What to do if the software you want isn’t in the official repos?

Or what if the official repos has an old version?

Some developers host their own repos...

(show Docker website)

Any risks of doing this?

Debian Free Software Guidelines

- Free redistribution.
- Inclusion of source code.
- Allowing for modifications and derived works.
- Integrity of the author's source code (as a compromise).
- No discrimination against persons or groups.
- No discrimination against fields of endeavor, like commercial use.
- The license needs to apply to all to whom the program is redistributed.
- License must not be specific to a product.
- License must not restrict other software.

https://www.debian.org/social_contract#guidelines

Demo: Package Creation

What just happened?

1. Code was written/compiled
2. Binaries, libraries, header files were copied to their appropriate paths mirroring where they should be on the system
   - Usually there is a `make install` that copies it correctly to the folder that you specified using `./configure`

Resources

- Developers shouldn't distribute their own software by Drew DeVault
- Rolling distribution releases versus periodic releases are a tradeoff by Chris Siebenmann

- Package Management by the DebianWiki

- pacman by ArchWiki
- Debian Packaging Tutorial. You can put this in your /usr/share/doc/ by installing packaging-tutorial
- Debian Binary Package Building HOWTO by Chr. Clemens Lee